ZOMATO

MINI PROJECT TEAM - 7

SLR- SLC REPORT

TEAM- 7 SLC-Mini-project-Part-I

Introduction:

This project deals with the data analysis of restaurant information in India. The project aims to explore hidden patterns that could affect the target variable - cost. The dataset is in CSV format and is cleaned with proper data types and missing values. The project is divided into three steps.

Problem:

Restaurants from all over the world can be found here in Bengaluru. From the United States to Japan, Russia to Antarctica, you get all types of cuisines here. Delivery, Dine-out, Pubs, Bars, Drinks, Buffet, Desserts, you name it and Bengaluru has it. Bengaluru is the best place for foodies. The number of restaurants is increasing day by day. Currently, it stands at approximately 12,000 restaurants. With such a high number of restaurants. This industry hasn't been saturated yet. And new restaurants are opening every day. However, it has become difficult for them to compete with already established restaurants. The key issues that continue to pose a challenge to them include high real estate costs, rising food costs, shortage of quality manpower, fragmented supply chain, and over-licensing.

Objective:

The newly started companies are not able to decide the cost that would happen per two people for once. So the Zomato company has a good analyst team who can predict the cost per two customers for one time so that the newly started restaurants and upcoming restaurants will be well prepared how the restaurant should invest in improving the ambiance and all other stuff to attract the customers. Assume you are the analyst team that Zomato has organized to help new and upcoming restaurants by letting them know the various reasons that customers look for and build a model which able to predict the cost for two people.

Understanding the Business Problem and Data Set:

The project analyses various aspects of a restaurant, such as its name, location, cuisines, and more. The dataset contains the following columns:

Name - Name of the restaurant - Object datatype

Online Order - Whether the customer ordered the menu online or not - Object datatype.

Book table - Whether the customer has booked the table or not - Object datatype

Rate - Rating of the restaurant that has by the customer - Numerical datatype

Votes - The votes given by the customer to the restaurant - Numerical datatype

Location - The city name where the restaurant is located - Object datatype

Rest Type - The type of restaurant - Object datatype

Dish liked - Dishes liked by the customer from the restaurant - Object datatype

Cuisines - The cuisines that have been prepared by the restaurant - Object datatype

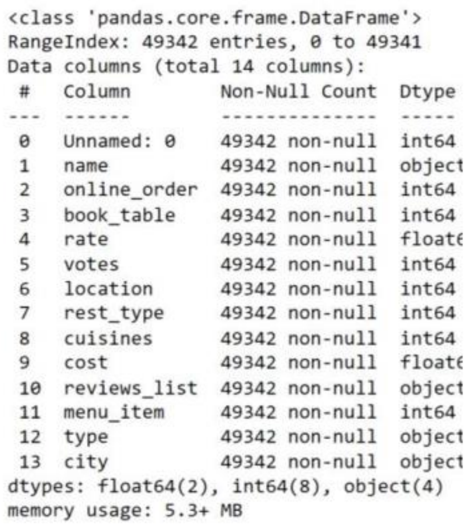
Approx Cost for two people - The approximate cost of the customer for two people - Numerical datatype

Reviews list - The reviews made by the customers on the restaurant - Object datatype

Menu Item - The menu items that are usually available at the restaurant - Object datatype

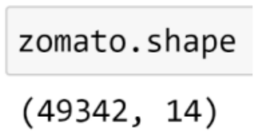
Listed in (type) - Contains the type of meal - Object datatype

Listed in (city) - This contains the neighbourhoods in which the restaurant is listed - Object datatype



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49342 entries, 0 to 49341
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Unnamed: 0    49342 non-null  int64
 1   name          49342 non-null  object
 2   online_order  49342 non-null  int64
 3   book_table    49342 non-null  int64
 4   rate          49342 non-null  float6
 5   votes         49342 non-null  int64
 6   location      49342 non-null  int64
 7   rest_type     49342 non-null  int64
 8   cuisines      49342 non-null  int64
 9   cost          49342 non-null  float6
 10  reviews_list  49342 non-null  object
 11  menu_item     49342 non-null  int64
 12  type          49342 non-null  object
 13  city          49342 non-null  object
dtypes: float64(2), int64(8), object(4)
memory usage: 5.3+ MB
```

We come to know that there are 2- Discrete series, 8- Continuous and 4- object datatype from the above.



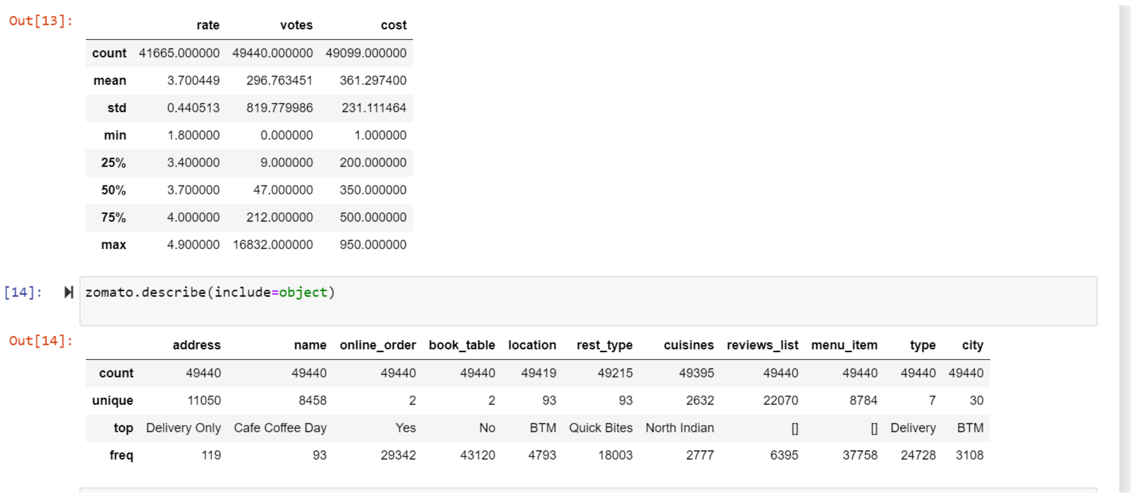
```
zomato.shape
```

```
(49342, 14)
```

We come to know that there are 49342 rows and 14 columns.

Data Cleaning:

The 5-point summary:


Out[13]:

	rate	votes	cost
count	41665.000000	49440.000000	49099.000000
mean	3.700449	296.763451	361.297400
std	0.440513	819.779986	231.111464
min	1.800000	0.000000	1.000000
25%	3.400000	9.000000	200.000000
50%	3.700000	47.000000	350.000000
75%	4.000000	212.000000	500.000000
max	4.900000	16832.000000	950.000000

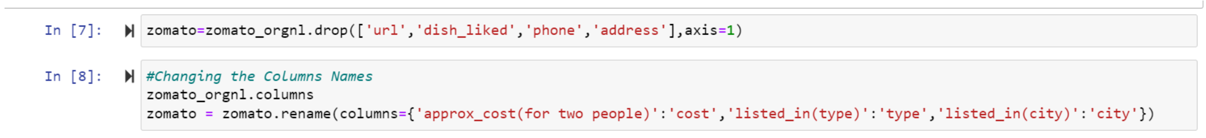
In [14]:
```
zomato.describe(include=object)
```
Out[14]:

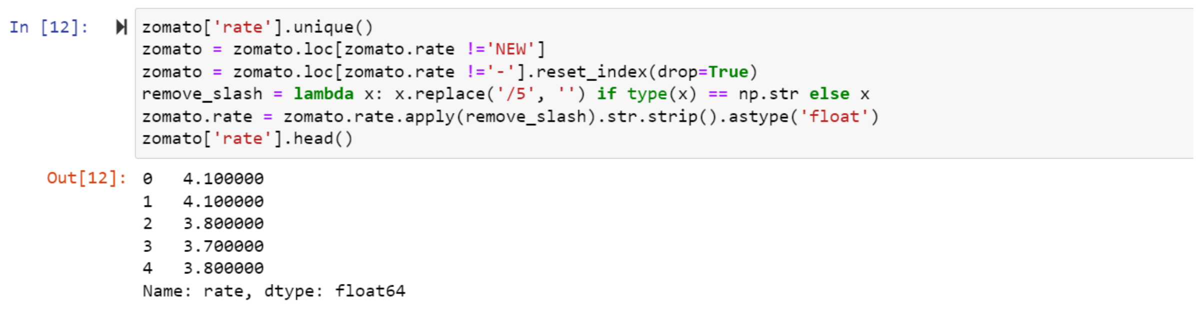
	address	name	online_order	book_table	location	rest_type	cuisines	reviews_list	menu_item	type	city
count	49440	49440	49440	49440	49419	49215	49395	49440	49440	49440	49440
unique	11050	8458	2	2	93	93	2632	22070	8784	7	30
top	Delivery Only	Cafe Coffee Day	Yes	No	BTM	Quick Bites	North Indian	[]	[]	Delivery	BTM
freq	119	93	29342	43120	4793	18003	2777	6395	37758	24728	3108

We could see the variation in the ‘votes’ 50%- 75%, which says that we have outliers there.

By getting the analysis from Object data, we come to know that which columns are to be dropped (columns which doesn’t contribute to project)

Filling the Null Values, Dropping unnecessary columns and Converting Datatype:

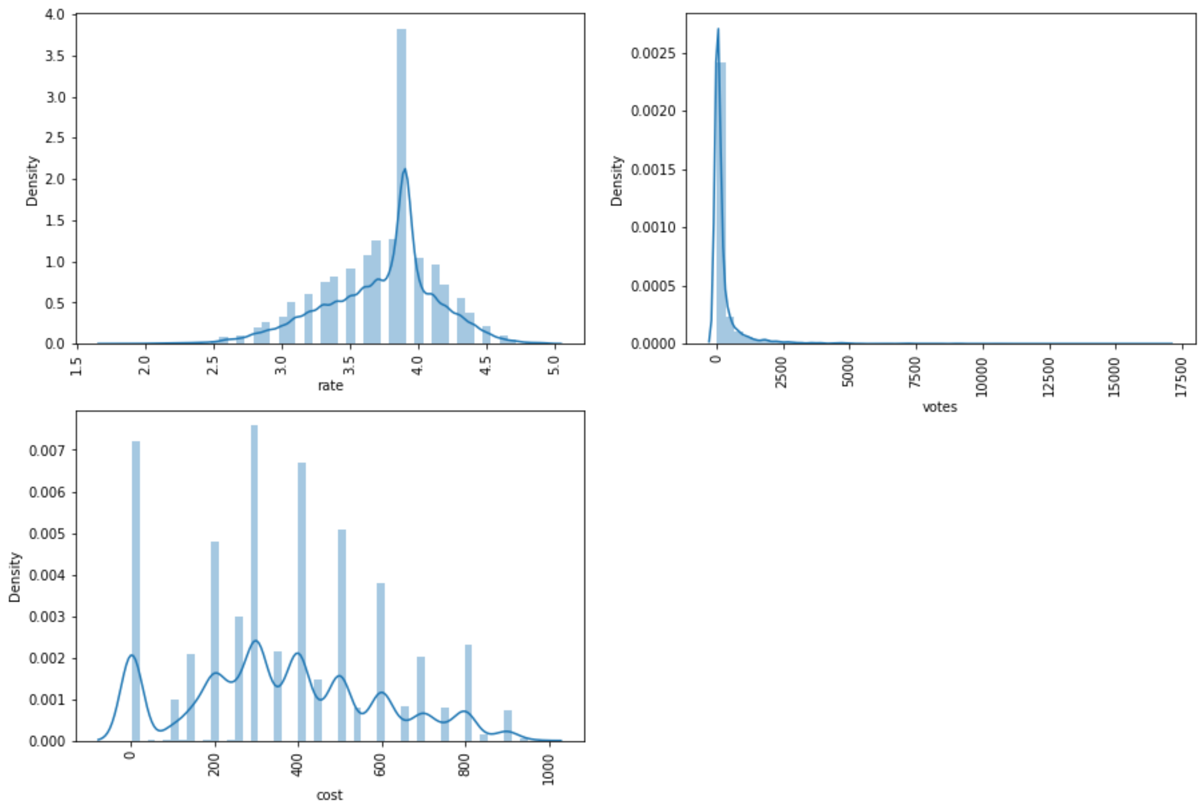

In [7]:
```
zomato=zomato_orgnl.drop(['url','dish_liked','phone','address'],axis=1)
```
In [8]:
```
#Changing the Columns Names
zomato_orgnl.columns
zomato = zomato.rename(columns={'approx_cost(for two people)':'cost','listed_in(type)':'type','listed_in(city)':'city'})
```


In [12]:
```
zomato['rate'].unique()
zomato = zomato.loc[zomato.rate !='NEW']
zomato = zomato.loc[zomato.rate !='-'].reset_index(drop=True)
remove_slash = lambda x: x.replace('/5', '') if type(x) == np.str else x
zomato.rate = zomato.rate.apply(remove_slash).str.strip().astype('float')
zomato['rate'].head()
```
Out[12]:
```
0    4.100000
1    4.100000
2    3.800000
3    3.700000
4    3.800000
Name: rate, dtype: float64
```

```
In [17]:  zomato['rate'].fillna(zomato['rate'].mode()[0],inplace=True)
           zomato['rest_type'].fillna(zomato['rest_type'].mode()[0],inplace=True)
           zomato['cost'].fillna(zomato['cost'].mean(),inplace=True)
```

Drop null values

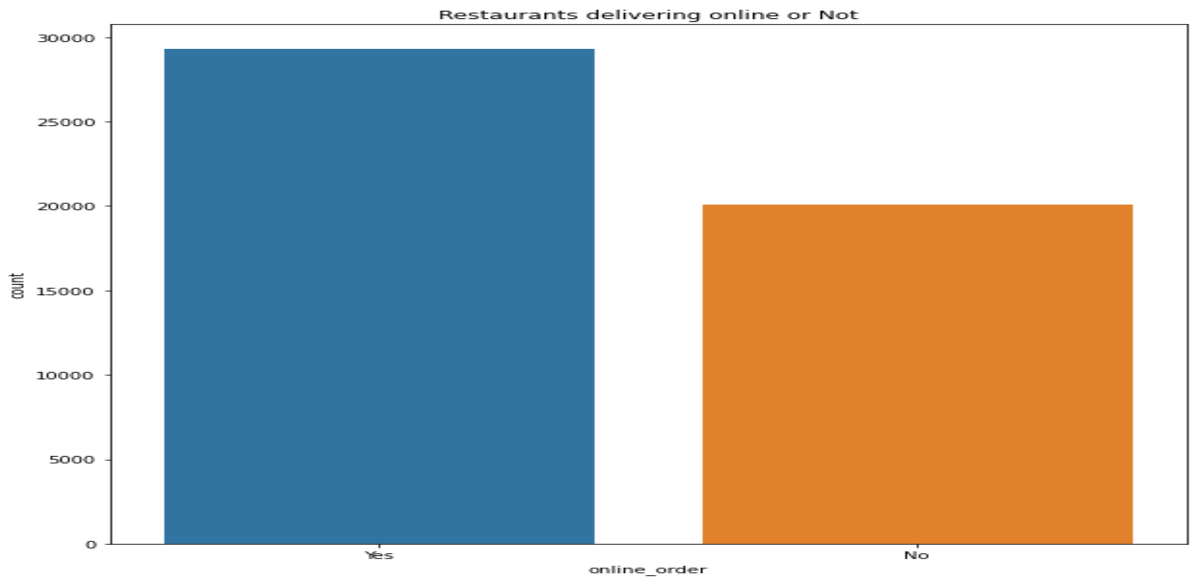
```
In [18]:  #Remove the NaN values from the dataset
           zomato.dropna(how='any',inplace=True)
           zomato.isnull().sum()

Out[18]:  address          0
           name             0
           online_order     0
           book_table       0
           rate             0
           votes            0
           location         0
           rest_type        0
           cuisines         0
           cost             0
           reviews_list     0
           menu_item        0
           type             0
           city             0
```

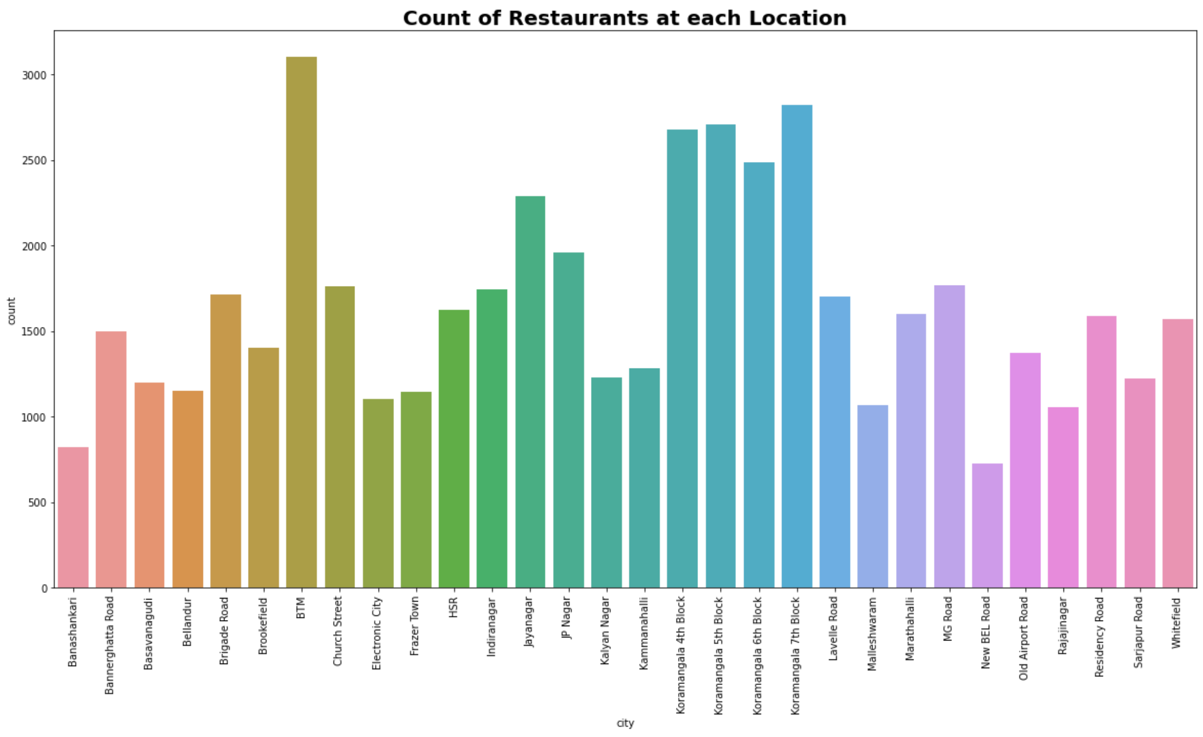


Checking Duplicates:

```
<Figure size 432x288 with 0 Axes>

In [21]:  #Removing the Duplicates
           zomato.duplicated().sum()
           zomato.drop_duplicates(inplace=True)

In [22]:  zomato.shape

Out[22]:  (49360, 14)
```

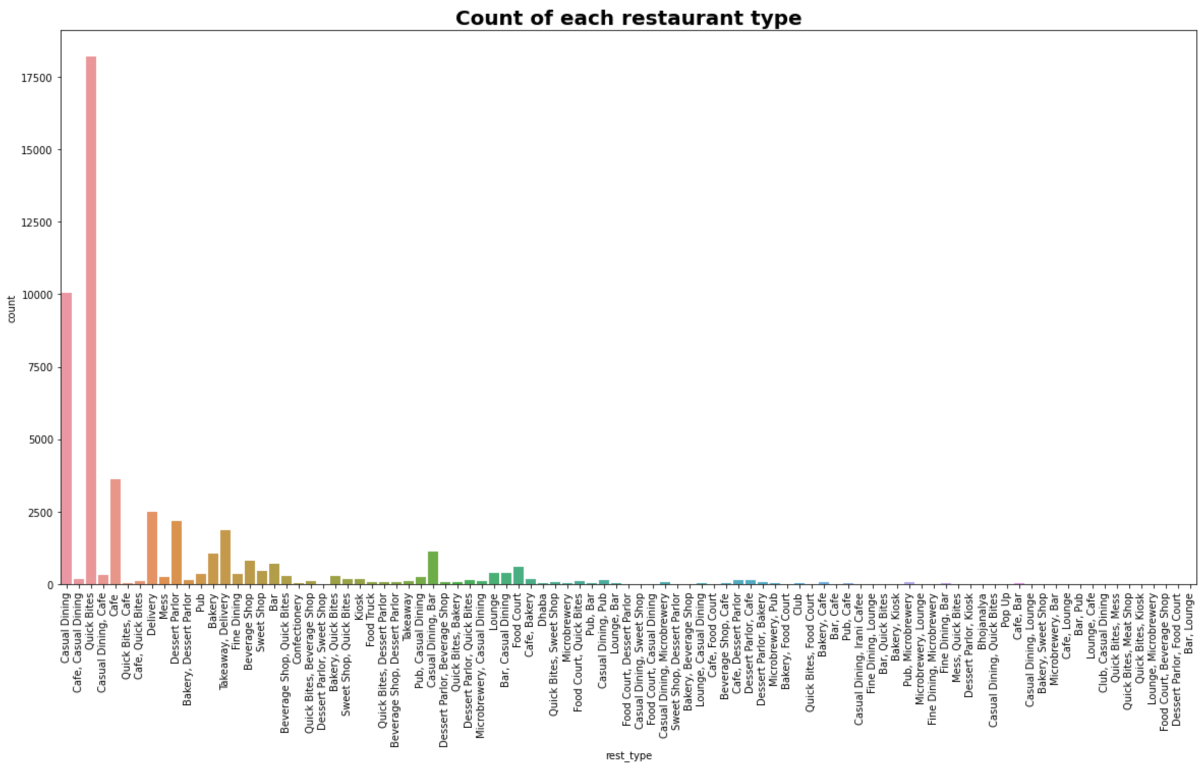
Graphs:

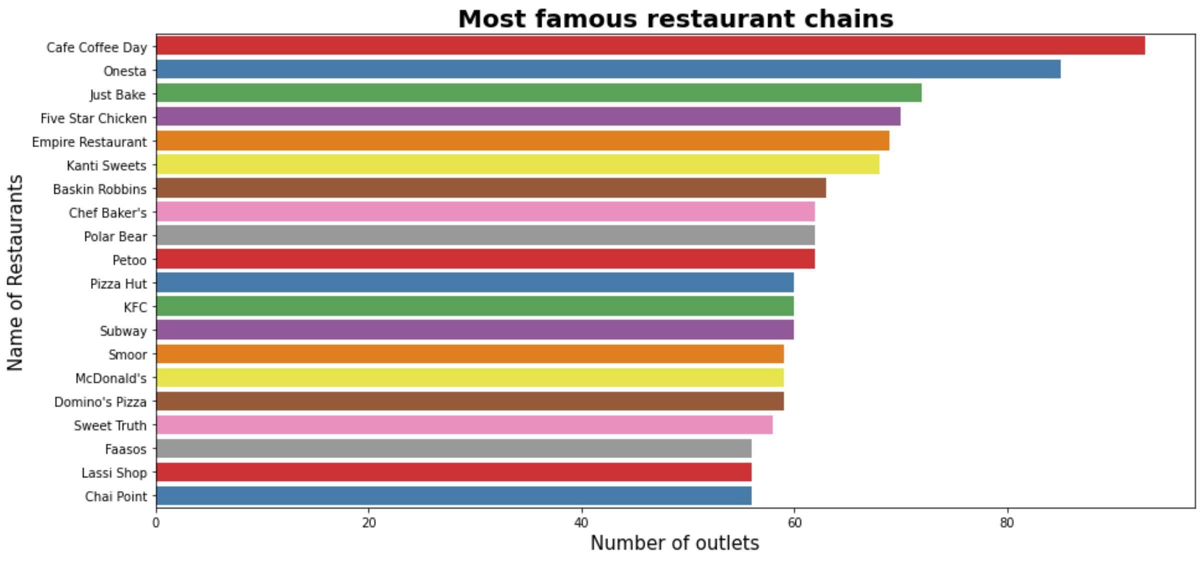


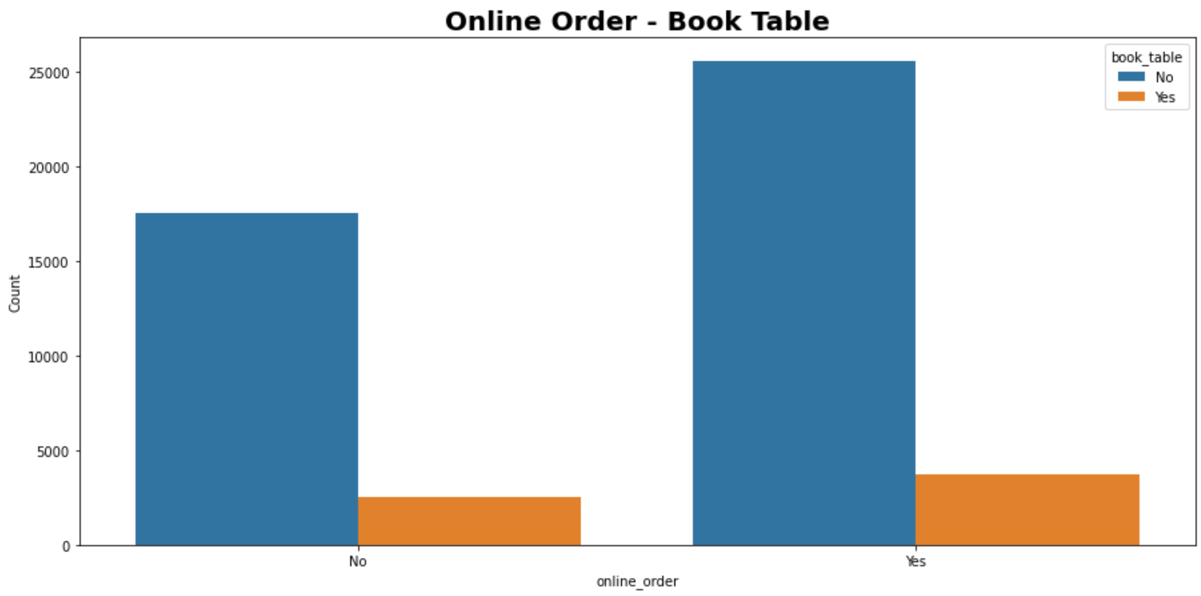


There are more than 3000 restaurants in BTM. Quick Google Search shows us that BTM is posh residential area so because of that there are quite a lot of restaurants and also it is famous for cafes. JP Nagar HSR, Koramangala 5th block, Whitefield, Indiranagar have more than 2000 restaurants and Jayanagar, Marathahalli, Bannerghatta Road have more than 1000 restaurants.

Restaurant types







Online Order - Book Table
Count
25000
20000
15000
10000
5000
0
No
Yes
online_order
book_table
No
Yes

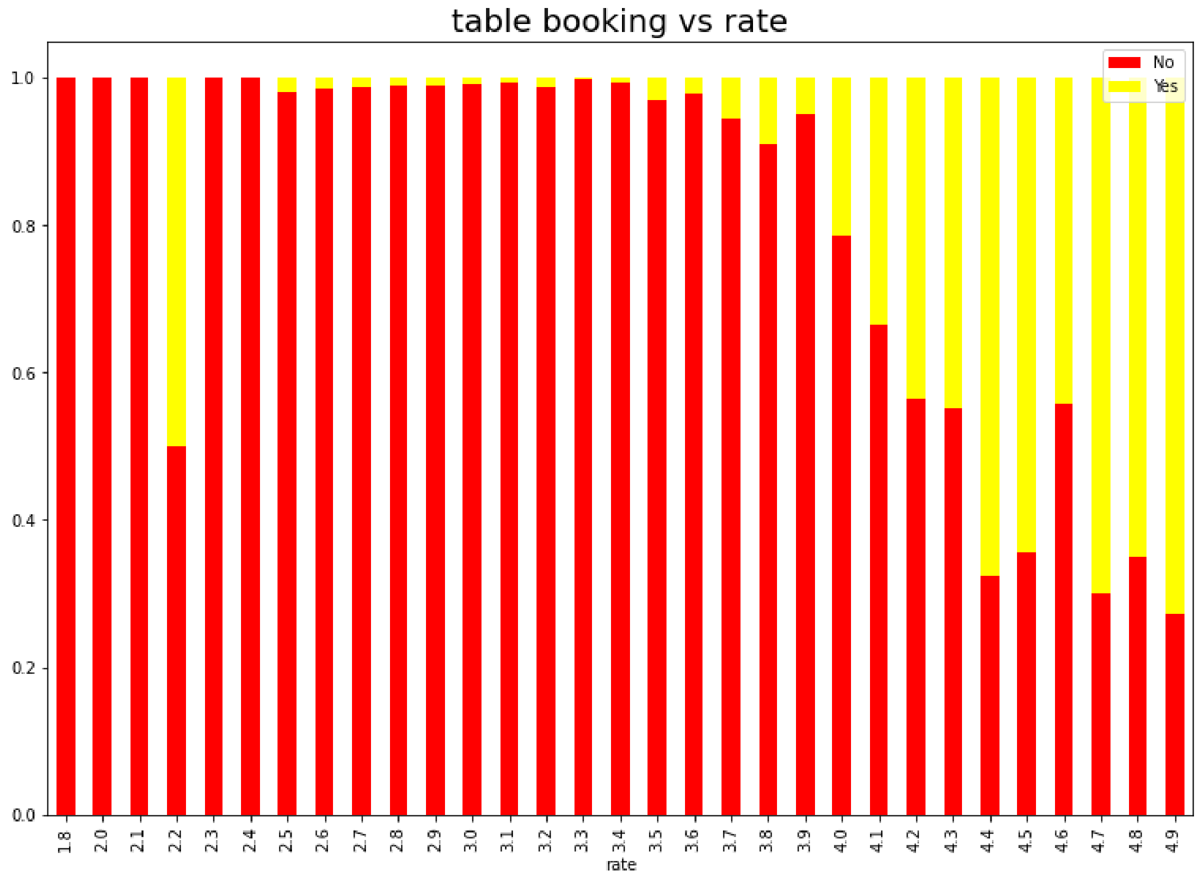
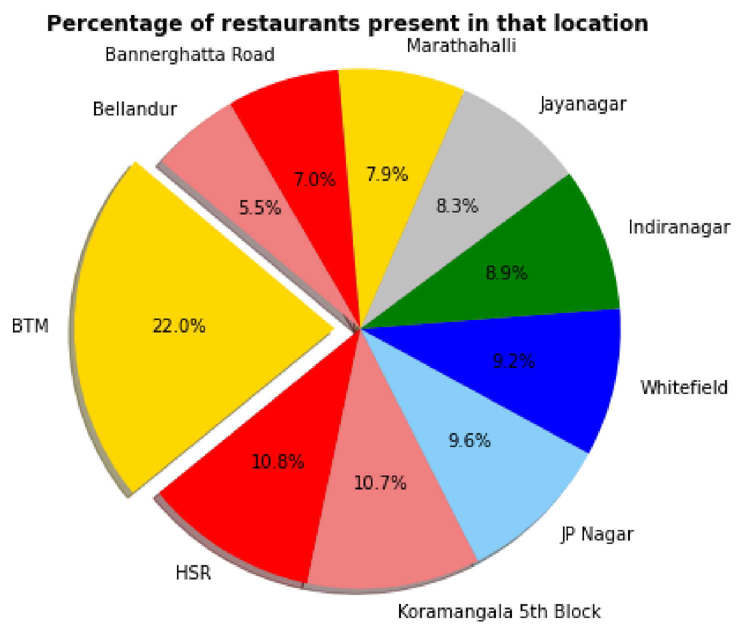
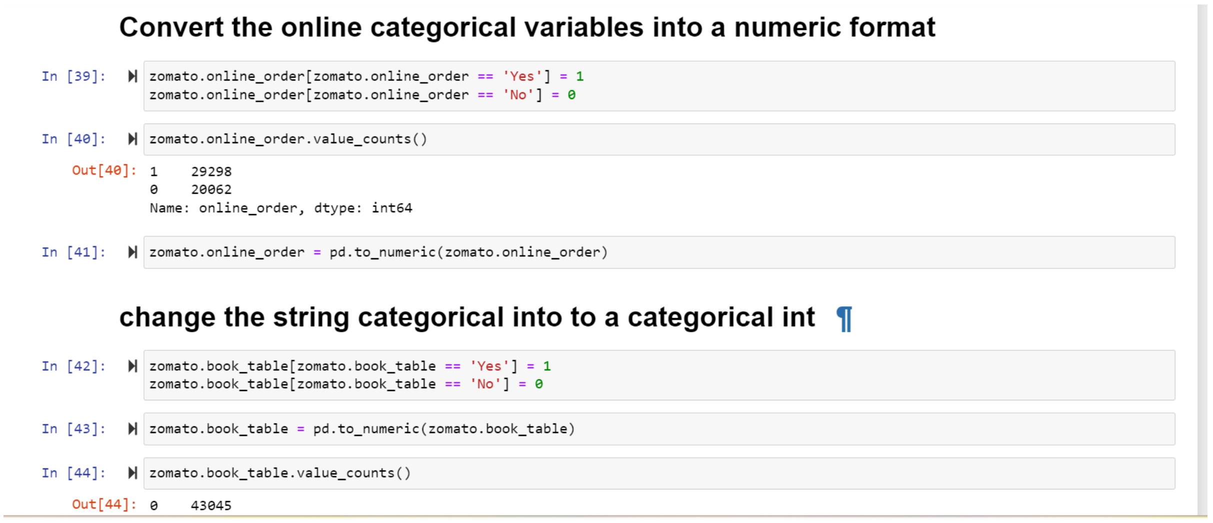


table booking vs rate
1.0
0.8
0.6
0.4
0.2
0.0
1.8
2.0
2.1
2.2
2.3
2.4
2.5
2.6
2.7
2.8
2.9
3.0
3.1
3.2
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4.0
4.1
4.2
4.3
4.4
4.5
4.6
4.7
4.8
4.9
rate
No
Yes



Encoding:



Convert the online categorical variables into a numeric format

```
In [39]: zomato.online_order[zomato.online_order == 'Yes'] = 1
         zomato.online_order[zomato.online_order == 'No'] = 0
```

```
In [40]: zomato.online_order.value_counts()
```

```
Out[40]: 1    29298
         0    20062
         Name: online_order, dtype: int64
```

```
In [41]: zomato.online_order = pd.to_numeric(zomato.online_order)
```

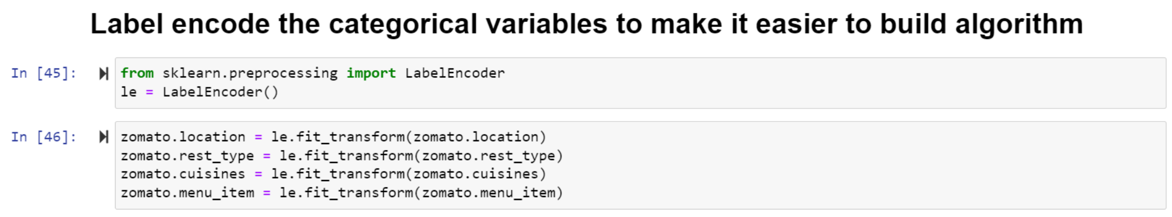
change the string categorical into to a categorical int ¶

```
In [42]: zomato.book_table[zomato.book_table == 'Yes'] = 1
         zomato.book_table[zomato.book_table == 'No'] = 0
```

```
In [43]: zomato.book_table = pd.to_numeric(zomato.book_table)
```

```
In [44]: zomato.book_table.value_counts()
```

```
Out[44]: 0    43045
```



Label encode the categorical variables to make it easier to build algorithm

```
In [45]: from sklearn.preprocessing import LabelEncoder
         le = LabelEncoder()
```

```
In [46]: zomato.location = le.fit_transform(zomato.location)
         zomato.rest_type = le.fit_transform(zomato.rest_type)
         zomato.cuisines = le.fit_transform(zomato.cuisines)
         zomato.menu_item = le.fit_transform(zomato.menu_item)
```

Perform statistical hypothesis testing on features to get an idea of whether features are impacting the target variables.

Dividing the dataset into categorical and numerical columns and then finding pvalue for each independent feature corresponding to dependent feture(cost). Keeping pvalue threshold as 0.05

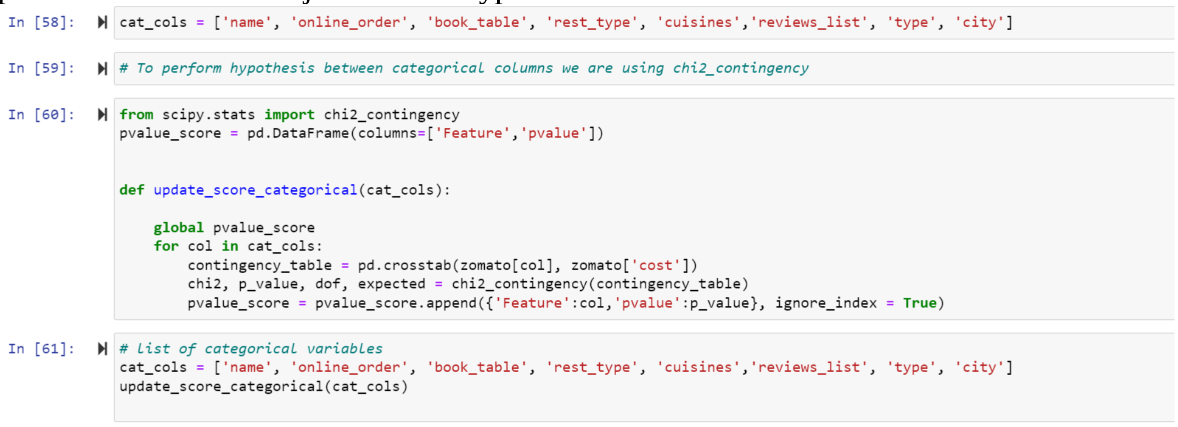
1. Hypothesis Testing for Categorical columns and stroke

Hypothesis:

H0: Independent variables are not significantly associated with the dependent variable (cost)

H1: Independent variables are significantly associated with the dependent variable (cost)

if p-value<0.05 we will reject the null hypothesis.



```
In [58]:  cat_cols = ['name', 'online_order', 'book_table', 'rest_type', 'cuisines','reviews_list', 'type', 'city']

In [59]:  # To perform hypothesis between categorical columns we are using chi2_contingency

In [60]:  from scipy.stats import chi2_contingency
          pvalue_score = pd.DataFrame(columns=['Feature','pvalue'])


          def update_score_categorical(cat_cols):

              global pvalue_score
              for col in cat_cols:
                  contingency_table = pd.crosstab(zomato[col], zomato['cost'])
                  chi2, p_value, dof, expected = chi2_contingency(contingency_table)
                  pvalue_score = pvalue_score.append({'Feature':col,'pvalue':p_value}, ignore_index = True)

In [61]:  # List of categorical variables
          cat_cols = ['name', 'online_order', 'book_table', 'rest_type', 'cuisines','reviews_list', 'type', 'city']
          update_score_categorical(cat_cols)
```

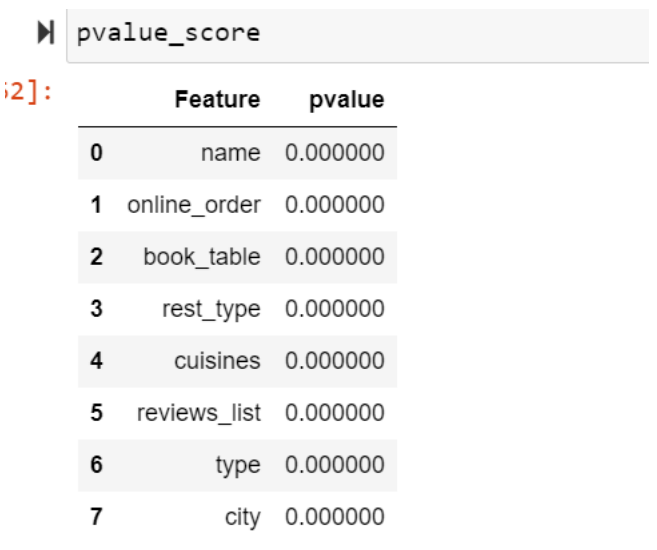
2. Hypothesis Testing for numerical columns and stroke

Hypothesis:

H0: The variables are not correlated with each other

H1: The variables are correlated with each other

if p-value<0.05 we will reject the null hypothesis.

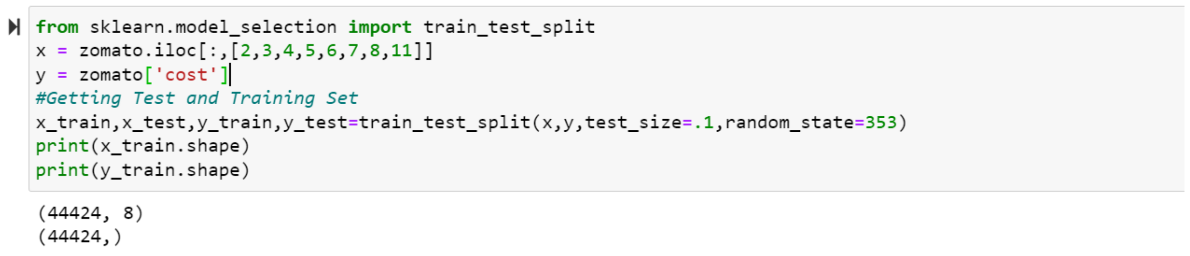


```
pvalue_score
```

2]:

	Feature	pvalue
0	name	0.000000
1	online_order	0.000000
2	book_table	0.000000
3	rest_type	0.000000
4	cuisines	0.000000
5	reviews_list	0.000000
6	type	0.000000
7	city	0.000000

Spliting the dataset into train and test data sets and Perform the scaling on both sets.

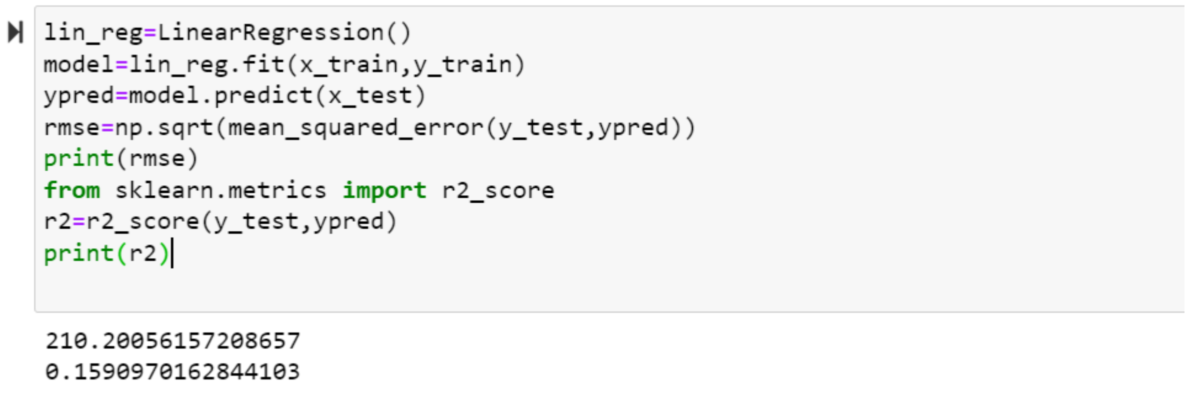


```
from sklearn.model_selection import train_test_split
x = zomato.iloc[:,[2,3,4,5,6,7,8,11]]
y = zomato['cost']
#Getting Test and Training Set
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.1,random_state=353)
print(x_train.shape)
print(y_train.shape)
```

```
(44424, 8)
(44424,)
```

Building the base model.

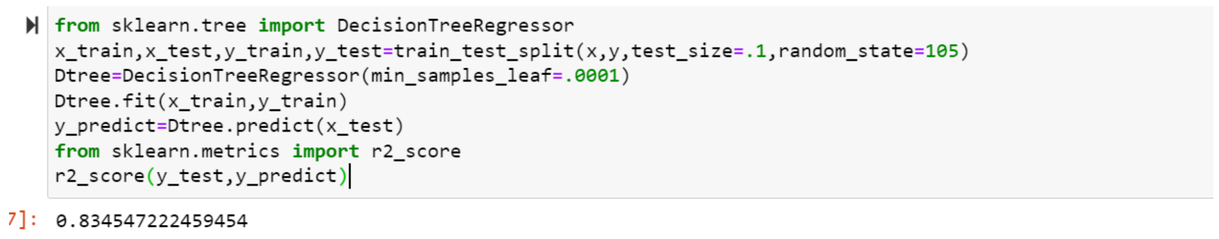
LinearRegression



```
lin_reg=LinearRegression()
model=lin_reg.fit(x_train,y_train)
ypred=model.predict(x_test)
rmse=np.sqrt(mean_squared_error(y_test,ypred))
print(rmse)
from sklearn.metrics import r2_score
r2=r2_score(y_test,ypred)
print(r2)
```

```
210.20056157208657
0.1590970162844103
```

Descision tree

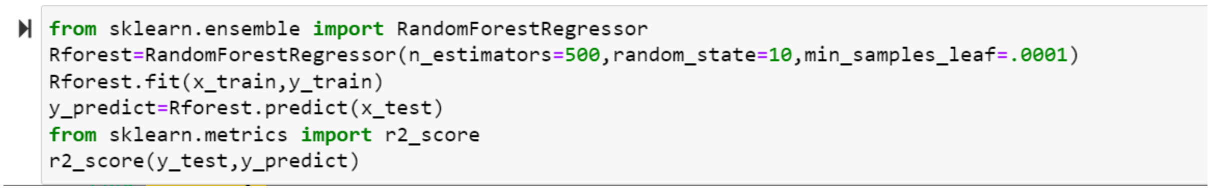


```
from sklearn.tree import DecisionTreeRegressor
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.1,random_state=105)
Dtree=DecisionTreeRegressor(min_samples_leaf=.0001)
Dtree.fit(x_train,y_train)
y_predict=Dtree.predict(x_test)
from sklearn.metrics import r2_score
r2_score(y_test,y_predict)
```

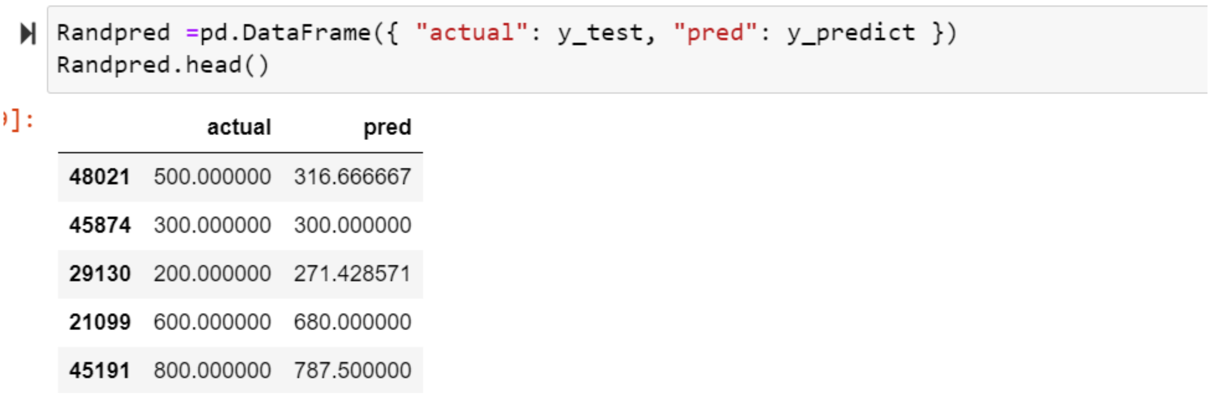
7]:

```
0.834547222459454
```

RandomForestRegressor



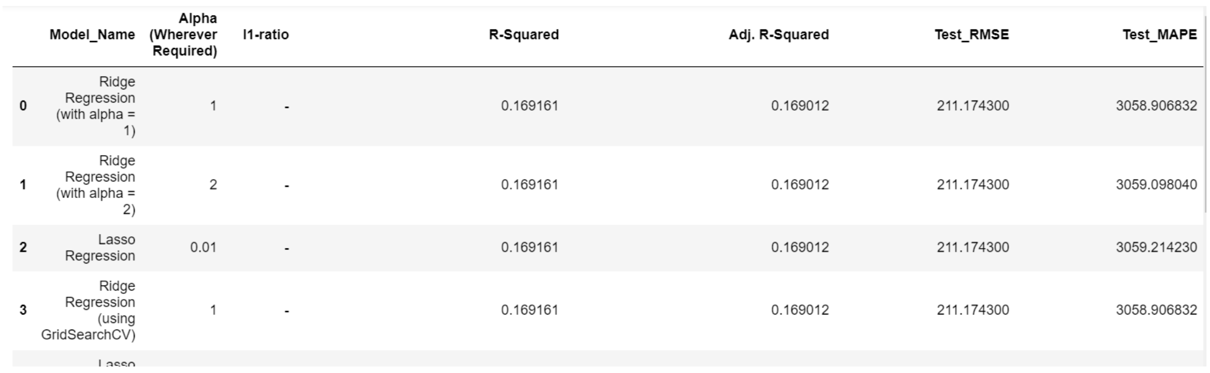
```
from sklearn.ensemble import RandomForestRegressor
Rforest=RandomForestRegressor(n_estimators=500,random_state=10,min_samples_leaf=.0001)
Rforest.fit(x_train,y_train)
y_predict=Rforest.predict(x_test)
from sklearn.metrics import r2_score
r2_score(y_test,y_predict)
```



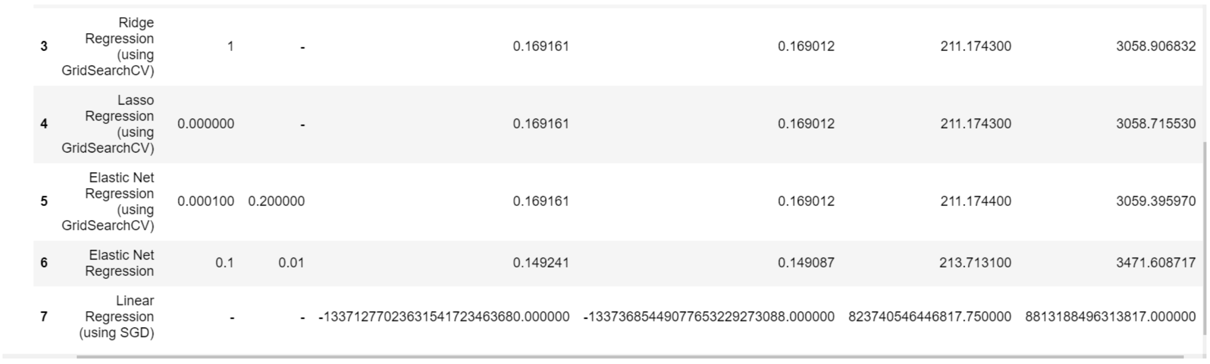
```
Randpred =pd.DataFrame({ "actual": y_test, "pred": y_predict })
Randpred.head()
```

	actual	pred
48021	500.000000	316.666667
45874	300.000000	300.000000
29130	200.000000	271.428571
21099	600.000000	680.000000
45191	800.000000	787.500000

Understanding the model’s performance, performing feature engineering again & feature selection.



	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Ridge Regression (with alpha = 1)	1	-	0.169161	0.169012	211.174300	3058.906832
1	Ridge Regression (with alpha = 2)	2	-	0.169161	0.169012	211.174300	3059.098040
2	Lasso Regression	0.01	-	0.169161	0.169012	211.174300	3059.214230
3	Ridge Regression (using GridSearchCV)	1	-	0.169161	0.169012	211.174300	3058.906832



3	Ridge Regression (using GridSearchCV)	1	-	0.169161	0.169012	211.174300	3058.906832
4	Lasso Regression (using GridSearchCV)	0.000000	-	0.169161	0.169012	211.174300	3058.715530
5	Elastic Net Regression (using GridSearchCV)	0.000100	0.200000	0.169161	0.169012	211.174400	3059.395970
6	Elastic Net Regression	0.1	0.01	0.149241	0.149087	213.713100	3471.608717
7	Linear Regression (using SGD)	-	-	-13371277023631541723463680.000000	-13373685449077653229273088.000000	823740546446817.750000	8813188496313817.000000

Spliting the data into train and test

```
sgd = SGDRegressor(random_state = 10)
linreg_with_SGD = sgd.fit(x_train, y_train)
print('RMSE on train set:', get_train_rmse(linreg_with_SGD))
print('RMSE on test set:', get_test_rmse(linreg_with_SGD))
```

```
RMSE on train set: 842466254405865.0
RMSE on test set: 823740546446817.8
```

```
ridge = Ridge(alpha = 1, max_iter = 500)
ridge.fit(x_train, y_train)
print('RMSE on test set:', get_test_rmse(ridge))
```

```
RMSE on test set: 211.1743
```

```
ridge = Ridge(alpha = 2, max_iter = 500)
ridge.fit(x_train, y_train)
print('RMSE on test set:', get_test_rmse(ridge))
```

```
RMSE on test set: 211.1743
```

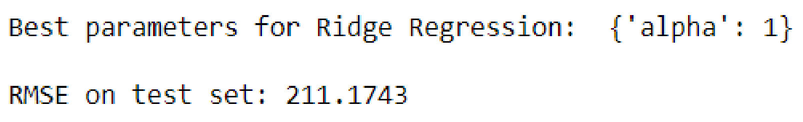
```
lasso = Lasso(alpha = 0.01, max_iter = 500)
lasso.fit(x_train, y_train)
print('RMSE on test set:', get_test_rmse(lasso))
```

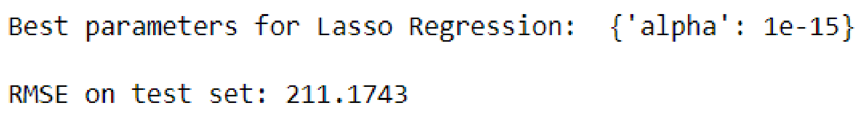
```
RMSE on test set: 211.1743
```

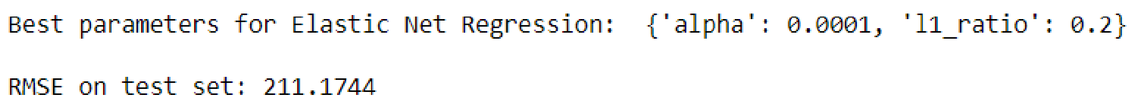
```
enet = ElasticNet(alpha = 0.1, l1_ratio = 0.01, max_iter = 500)
enet.fit(x_train, y_train)
print('RMSE on test set:', get_test_rmse(enet))
```

```
RMSE on test set: 213.7131
```

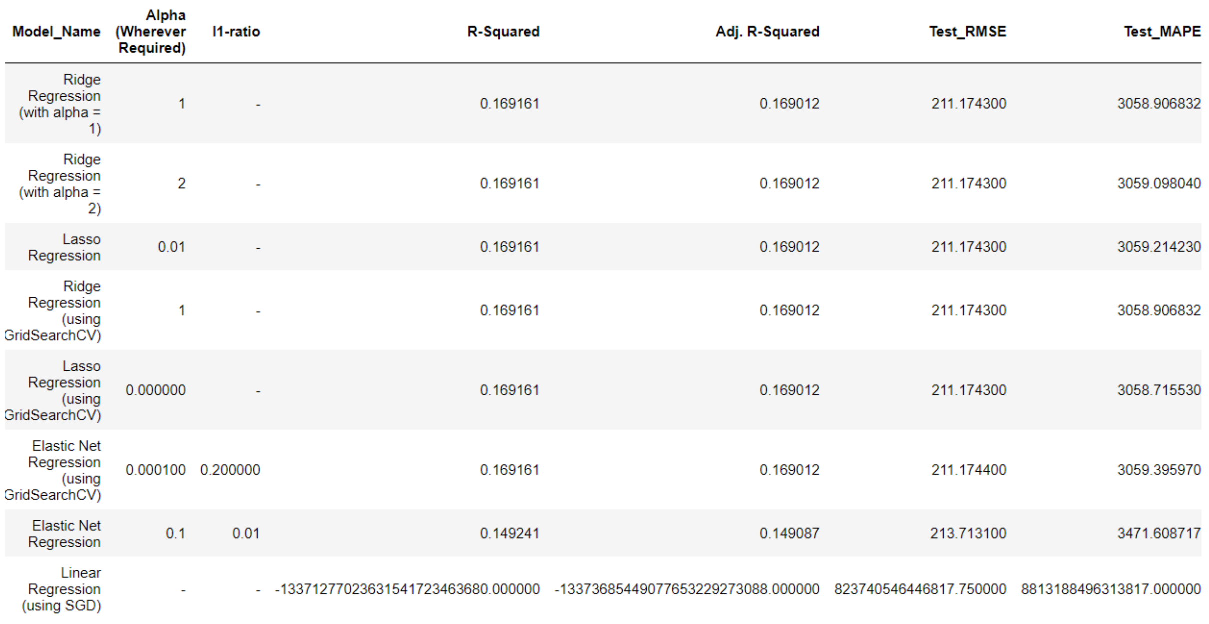
Outputs of Hyper parameter tuning with various models


```
Best parameters for Ridge Regression:  {'alpha': 1}

RMSE on test set: 211.1743
```


```
Best parameters for Lasso Regression:  {'alpha': 1e-15}

RMSE on test set: 211.1743
```


```
Best parameters for Elastic Net Regression:  {'alpha': 0.0001, 'l1_ratio': 0.2}

RMSE on test set: 211.1744
```

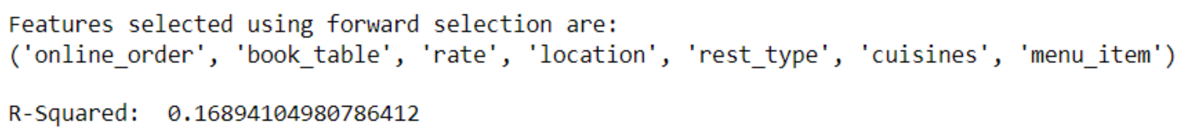
Score card of various models built



Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
Ridge Regression (with alpha = 1)	1	-	0.169161	0.169012	211.174300	3058.906832
Ridge Regression (with alpha = 2)	2	-	0.169161	0.169012	211.174300	3059.098040
Lasso Regression	0.01	-	0.169161	0.169012	211.174300	3059.214230
Ridge Regression (using GridSearchCV)	1	-	0.169161	0.169012	211.174300	3058.906832
Lasso Regression (using GridSearchCV)	0.000000	-	0.169161	0.169012	211.174300	3058.715530
Elastic Net Regression (using GridSearchCV)	0.000100	0.200000	0.169161	0.169012	211.174400	3059.395970
Elastic Net Regression	0.1	0.01	0.149241	0.149087	213.713100	3471.608717
Linear Regression (using SGD)	-	-	-1337127702363154172346368 0.000000	-1337368544907765322927308 8.000000	823740546446817.750000	8813188496313817.000000

Inference: We can see that Lasso Regression (using GridSearchCV) has the lowest test RMSE and test MAPE.

Feature selection using forward selection method


```
Features selected using forward selection are:
('online_order', 'book_table', 'rate', 'location', 'rest_type', 'cuisines', 'menu_item')

R-Squared:  0.16894104980786412
```

TEAM- 7 SLC-Mini-project-Part-II

Introduction:

This project deals with the data analysis of restaurant information in India. The project aims to explore hidden patterns that could affect the target variable - online order. The dataset is in CSV format and is cleaned with proper data types and missing values. The project is divided into three steps.

Understanding the Business Problem:

The project analyses various aspects of a restaurant, such as its name, location, cuisines, and more. The dataset contains the following columns:

Name - Name of the restaurant - Object datatype

Online Order - Whether the customer ordered the menu online or not - Object datatype.

Book table - Whether the customer has booked the table or not - Object datatype

Rate - Rating of the restaurant that has by the customer - Numerical datatype

Votes - The votes given by the customer to the restaurant - Numerical datatype

Location - The city name where the restaurant is located - Object datatype

Rest Type - The type of restaurant - Object datatype

Dish liked - Dishes liked by the customer from the restaurant - Object datatype

Cuisines - The cuisines that have been prepared by the restaurant - Object datatype

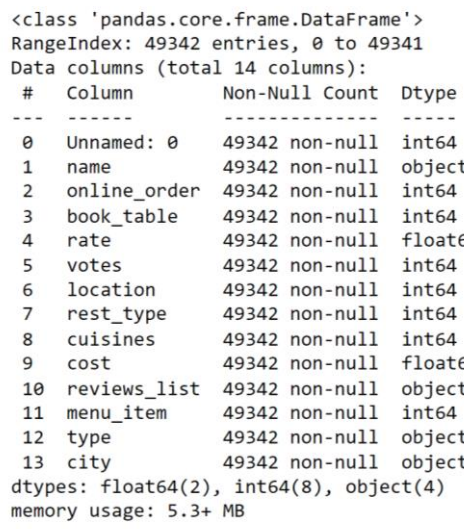
Approx Cost for two people - The approximate cost of the customer for two people - Numerical datatype

Reviews list - The reviews made by the customers on the restaurant - Object datatype

Menu Item - The menu items that are usually available at the restaurant - Object datatype

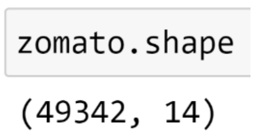
Listed in (type) - Contains the type of meal - Object datatype

Listed in (city) - This contains the neighbourhoods in which the restaurant is listed - Object datatype



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49342 entries, 0 to 49341
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Unnamed: 0    49342 non-null  int64
 1   name          49342 non-null  object
 2   online_order  49342 non-null  int64
 3   book_table    49342 non-null  int64
 4   rate          49342 non-null  float6
 5   votes         49342 non-null  int64
 6   location      49342 non-null  int64
 7   rest_type     49342 non-null  int64
 8   cuisines      49342 non-null  int64
 9   cost          49342 non-null  float6
 10  reviews_list  49342 non-null  object
 11  menu_item     49342 non-null  int64
 12  type          49342 non-null  object
 13  city          49342 non-null  object
dtypes: float64(2), int64(8), object(4)
memory usage: 5.3+ MB
```

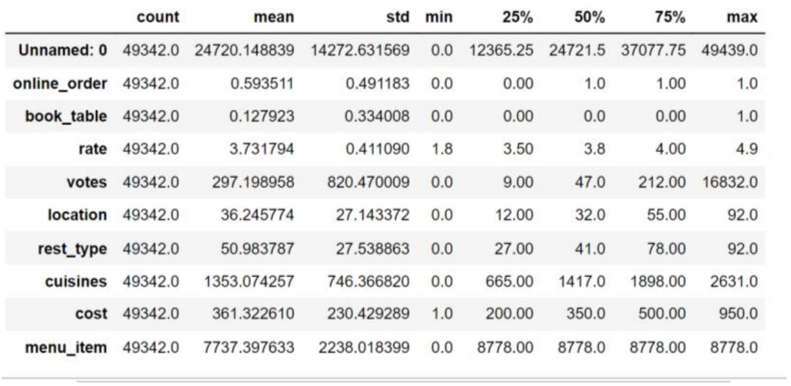
We come to know that there are 2- Discrete series, 8- Continuous and 4- object datatype from the above.



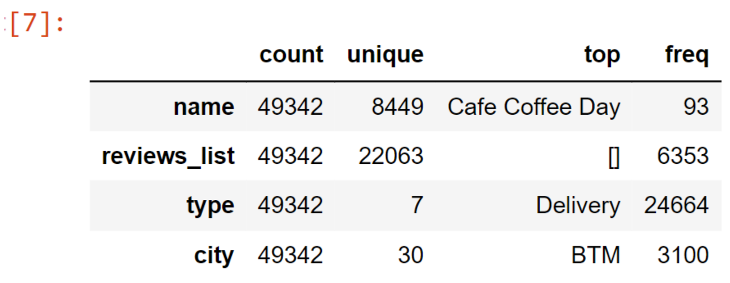
```
zomato.shape
```

```
(49342, 14)
```

We come to know that there are 49342 rows and 14 columns.



	count	mean	std	min	25%	50%	75%	max
Unnamed: 0	49342.0	24720.148839	14272.631569	0.0	12365.25	24721.5	37077.75	49439.0
online_order	49342.0	0.593511	0.491183	0.0	0.00	1.0	1.00	1.0
book_table	49342.0	0.127923	0.334008	0.0	0.00	0.0	0.00	1.0
rate	49342.0	3.731794	0.411090	1.8	3.50	3.8	4.00	4.9
votes	49342.0	297.198958	820.470009	0.0	9.00	47.0	212.00	16832.0
location	49342.0	36.245774	27.143372	0.0	12.00	32.0	55.00	92.0
rest_type	49342.0	50.983787	27.538863	0.0	27.00	41.0	78.00	92.0
cuisines	49342.0	1353.074257	746.366820	0.0	665.00	1417.0	1898.00	2631.0
cost	49342.0	361.322610	230.429289	1.0	200.00	350.0	500.00	950.0
menu_item	49342.0	7737.397633	2238.018399	0.0	8778.00	8778.0	8778.00	8778.0



:[7]:

	count	unique	top	freq
name	49342	8449	Cafe Coffee Day	93
reviews_list	49342	22063	[]	6353
type	49342	7	Delivery	24664
city	49342	30	BTM	3100

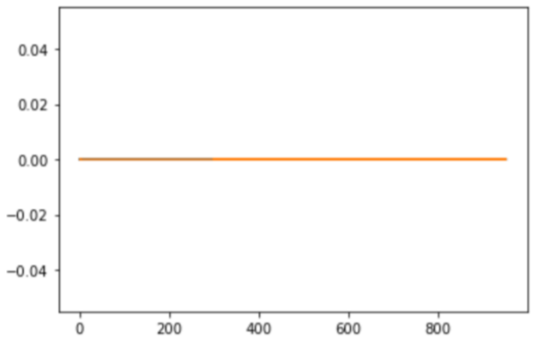
We come to know that there is no missing value in the given dataset, as it is pre-processed in part I.

Data Analysis:

The project analyses the dataset and finds hidden patterns that could affect the target variable - online order. The analysis is divided into two parts: Univariate Analysis and Bivariate Analysis.

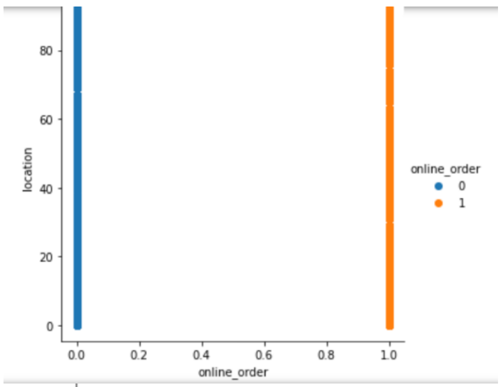
Univariate Analysis:

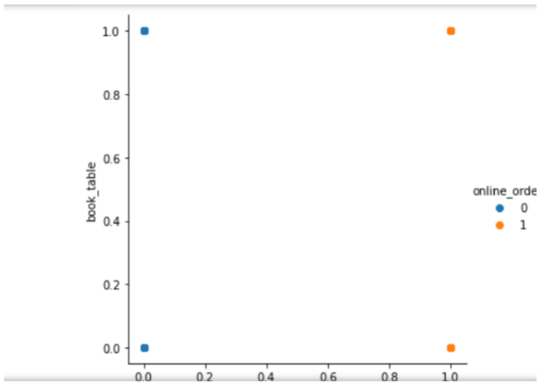
The univariate analysis analyses the variables individually. The project plotted the cost of online order vs. no online order to see the distribution of the target variable.

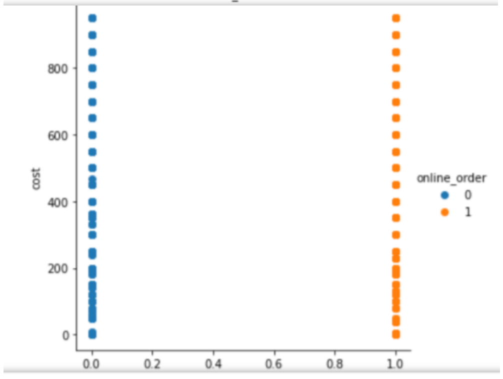


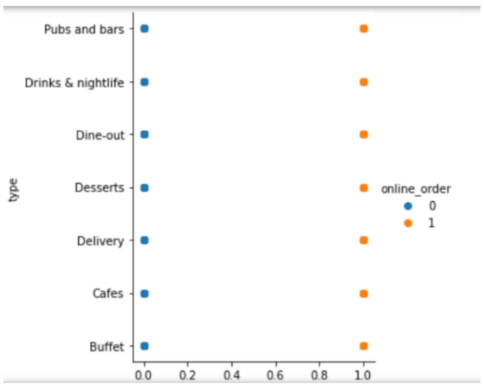
Bivariate Analysis:

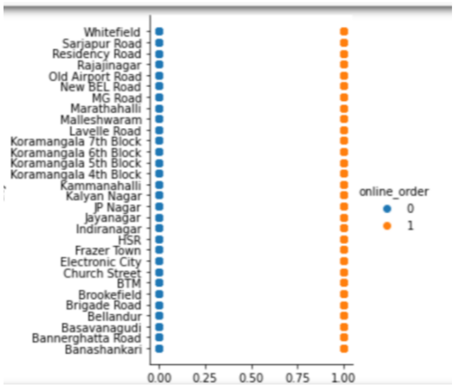
The bivariate analysis analyses the relationship between two variables. The project visualized the relationship between the target variable and important features such as location, book table, type, city, and cost.











Multivariate Analysis:

A Pair plot is plotted to find the corelation between various variables in the dataset, We can't derive a conclusion or any correlations.

Feature Selection:

The project used various techniques to select the most relevant features for the target variable - online order. The techniques include:

i) Identify the categorical and numerical features

ii) Check the distribution of the target variable

iii) Plot the correlation matrix

iv) Visualize the relationship between the target variable and important features

v) Perform t-test to compare "online_order" rate between restaurants with and without table booking

vi) Perform Lasso regression to select the most relevant features

vii) Perform Random Forest classification to determine feature importance

i) Identify the categorical and numerical features:

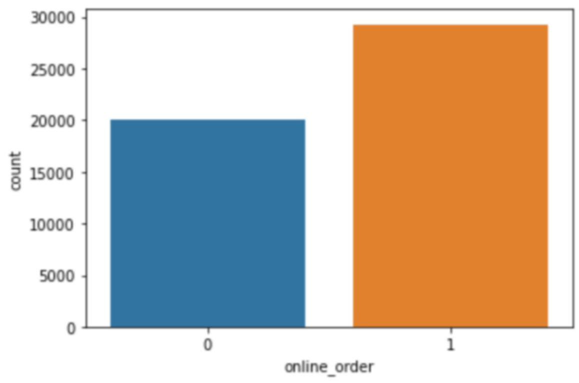
In this step, the project identifies the categorical and numerical features in the dataset. This is important because different types of features require different pre-processing techniques and modelling approaches.

Here, the categorical features are 'location', 'rest_type', 'cuisines', 'menu_item', and 'type', and the numerical features are 'votes', 'approx_cost(for two people)', 'rating', and 'listed_in(type)'. The project creates two separate dataframes for the categorical and numerical features respectively, to facilitate their pre-processing and modelling.

ii) Check the distribution of the target variable:

In this step, the project checks the distribution of the target variable, which is 'online_order'. This is important to understand the distribution of the target variable, which helps in choosing an appropriate evaluation metric and modeling approach.

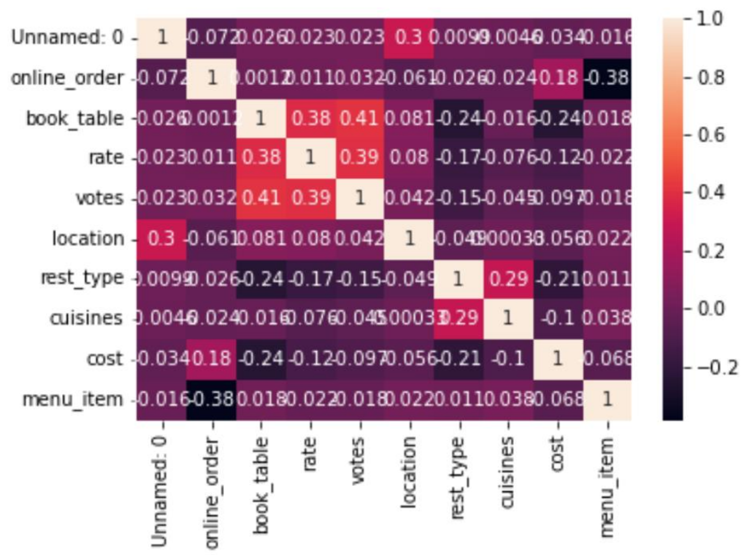
Here, the project uses the 'value_counts' method to count the number of occurrences of each unique value of 'online_order', and then sets the 'normalize' parameter to 'True' to obtain the proportion of each unique value in the dataset. This helps to understand the distribution of the target variable, which is 56% 'Yes' and 44% 'No'.



iii) Plot the correlation matrix:

In this step, the project plots a correlation matrix to visualize the pairwise correlations between the numerical features and the target variable. This helps to identify the features that are strongly correlated with the target variable.

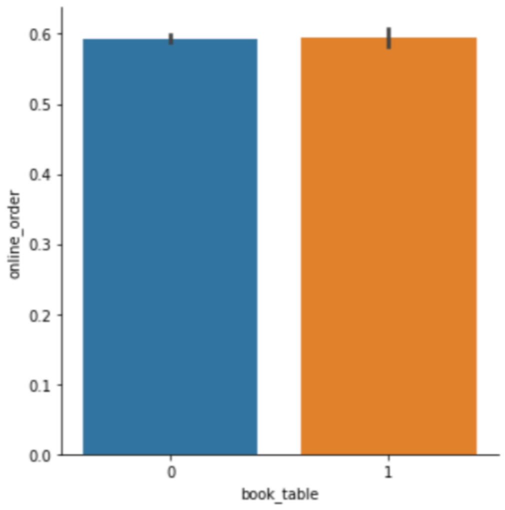
Here, the project uses the 'corr' method to calculate the pairwise correlations between the numerical features, and then uses the 'heatmap' function from the 'seaborn' library to plot the correlation matrix. The correlation matrix shows that 'votes' and 'rating' are strongly correlated with 'online_order', whereas 'approx_cost (for two people)' and 'listed_in(type)' have weak correlations

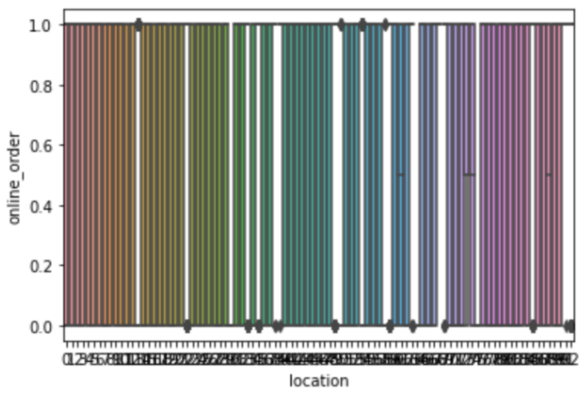


iv) Visualize the relationship between the target variable and important features:

In this step, the project visualizes the relationship between the target variable and important features using various plots, such as boxplots and barplots. This helps to understand the relationship between the target variable and the important features, and to identify any patterns or trends.

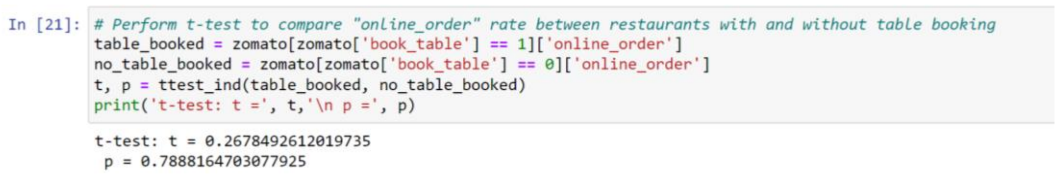
Here, the project uses the 'boxplot' function from the 'seaborn' library to plot a boxplot of 'votes' for each unique value of 'online_order', and uses the 'barplot' function to plot a barplot of 'rating' for each unique value of 'online_order'. These plots show that restaurants with online ordering tend to have higher ratings and more votes.





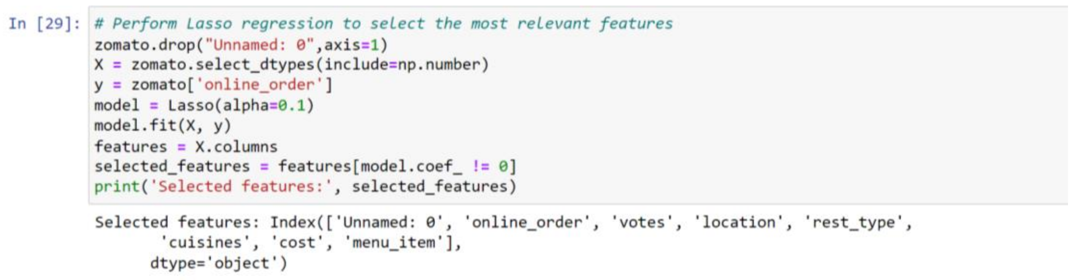
v) Perform t-test to compare "online_order" rate between restaurants with and without table booking:

In this step, the project performs a t-test to compare the rate of 'online_order' between restaurants with and without table booking. This helps to identify if there is a significant difference in the rate of online ordering between the two.



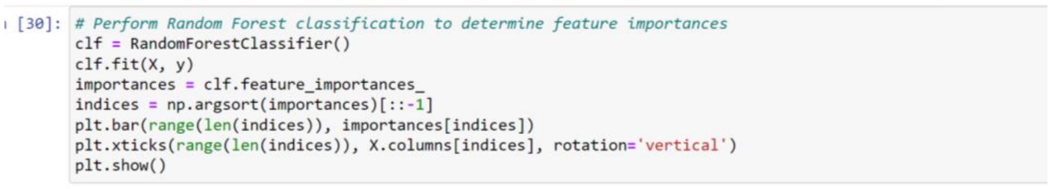
```
In [21]: # Perform t-test to compare "online_order" rate between restaurants with and without table booking
         table_booked = zomato[zomato['book_table'] == 1]['online_order']
         no_table_booked = zomato[zomato['book_table'] == 0]['online_order']
         t, p = ttest_ind(table_booked, no_table_booked)
         print('t-test: t =', t,'\n p =', p)

         t-test: t = 0.2678492612019735
          p = 0.7888164703077925
```

vi) Perform Lasso regression to select the most relevant features

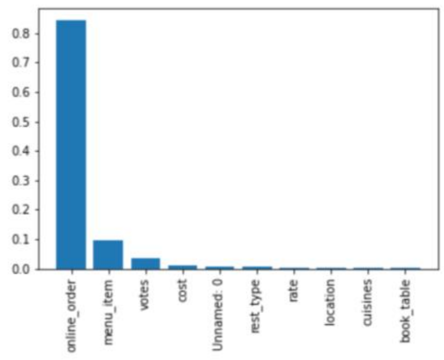


```
In [29]: # Perform Lasso regression to select the most relevant features
         zomato.drop("Unnamed: 0",axis=1)
         X = zomato.select_dtypes(include=np.number)
         y = zomato['online_order']
         model = Lasso(alpha=0.1)
         model.fit(X, y)
         features = X.columns
         selected_features = features[model.coef_ != 0]
         print('Selected features:', selected_features)

         Selected features: Index(['Unnamed: 0', 'online_order', 'votes', 'location', 'rest_type',
                'cuisines', 'cost', 'menu_item'],
               dtype='object')
```

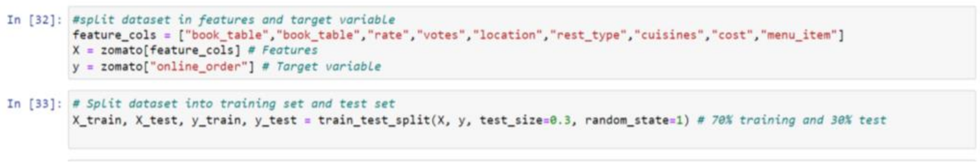
vii) Perform Random Forest classification to determine feature importance



```
[30]: # Perform Random Forest classification to determine feature importances
      clf = RandomForestClassifier()
      clf.fit(X, y)
      importances = clf.feature_importances_
      indices = np.argsort(importances)[::-1]
      plt.bar(range(len(indices)), importances[indices])
      plt.xticks(range(len(indices)), X.columns[indices], rotation='vertical')
      plt.show()
```



Split the data into train and test sets.



```
In [32]: #split dataset in features and target variable
         feature_cols = ["book_table","book_table","rate","votes","location","rest_type","cuisines","cost","menu_item"]
         X = zomato[feature_cols] # Features
         y = zomato["online_order"] # Target variable

In [33]: # Split dataset into training set and test set
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30% test
```

Building the base model and identifying the metric based on the problem

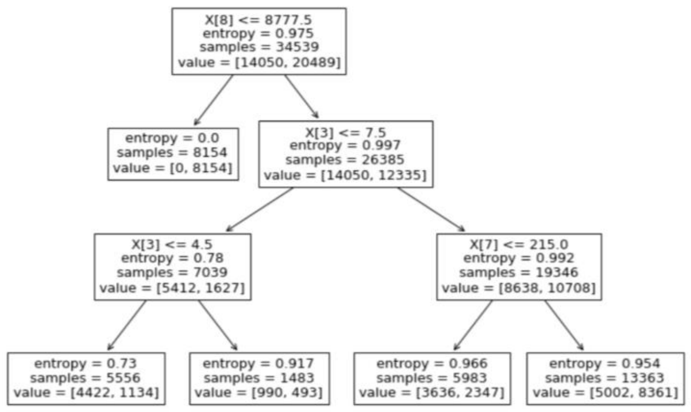
```
In [34]: # Create Decision Tree classifer object
         clf = DecisionTreeClassifier()

         # Train Decision Tree Classifer
         clf = clf.fit(X_train,y_train)

         #Predict the response for test dataset
         y_pred = clf.predict(X_test)
```

```
In [35]: # Model Accuracy, how often is the classifier correct?
         print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

         Accuracy: 0.9430520840370196
```

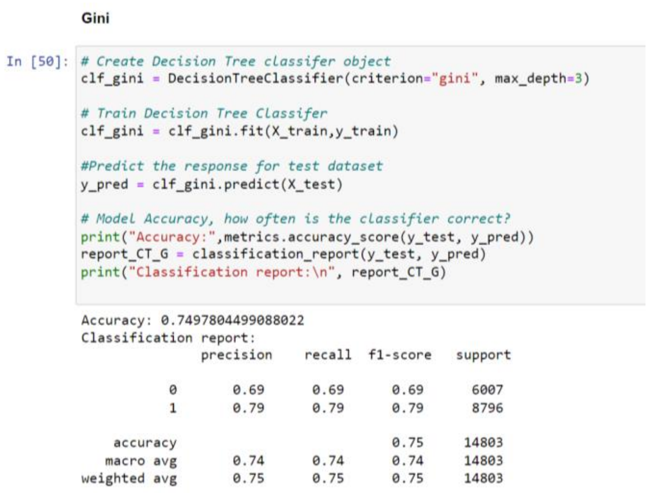
```
In [36]: from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, y_pred)
         print('Confusion matrix\n\n', cm)

         Confusion matrix

          [[5632  375]
          [ 468 8328]]
```

Optimizing decision tree

```
In [51]: # Create Decision Tree classifer object
         clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

         # Train Decision Tree Classifer
         clf = clf.fit(X_train,y_train)

         #Predict the response for test dataset
         y_pred = clf.predict(X_test)

         # Model Accuracy, how often is the classifier correct?
         print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
         report_CT_ENT = classification_report(y_test, y_pred)
         print("Classification report:\n", report_CT_ENT)


         Accuracy: 0.7384989529149497
         Classification report:
                        precision    recall  f1-score   support

                    0       0.69      0.66      0.67      6007
                    1       0.77      0.80      0.78      8796

             accuracy                           0.74     14803
            macro avg       0.73      0.73      0.73     14803
         weighted avg       0.74      0.74      0.74     14803
```



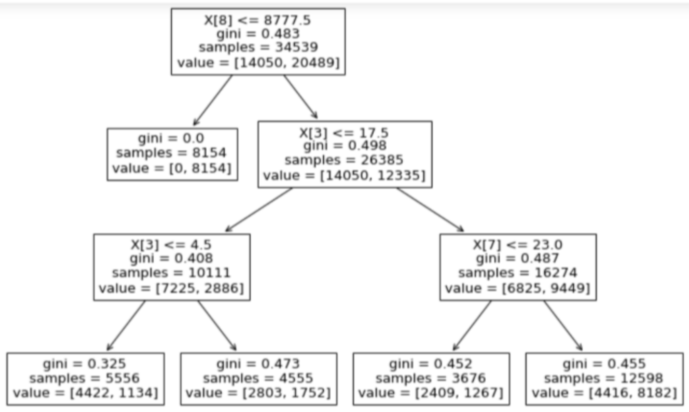
Improving the metric by trying different models, changing the feature engineering methods, feature selection, etc.



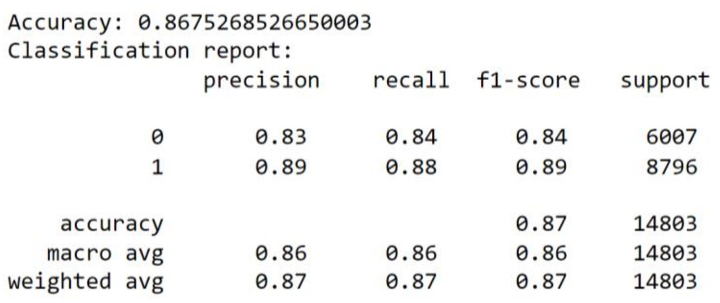
Gini

```
In [50]: # Create Decision Tree classifer object
         clf_gini = DecisionTreeClassifier(criterion="gini", max_depth=3)

         # Train Decision Tree Classifer
         clf_gini = clf_gini.fit(X_train,y_train)

         #Predict the response for test dataset
         y_pred = clf_gini.predict(X_test)

         # Model Accuracy, how often is the classifier correct?
         print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
         report_CT_G = classification_report(y_test, y_pred)
         print("Classification report:\n", report_CT_G)
```

```
Accuracy: 0.7497804499088022
Classification report:
               precision    recall  f1-score   support

           0       0.69      0.69      0.69      6007
           1       0.79      0.79      0.79      8796

    accuracy                           0.75     14803
   macro avg       0.74      0.74      0.74     14803
weighted avg       0.75      0.75      0.75     14803
```

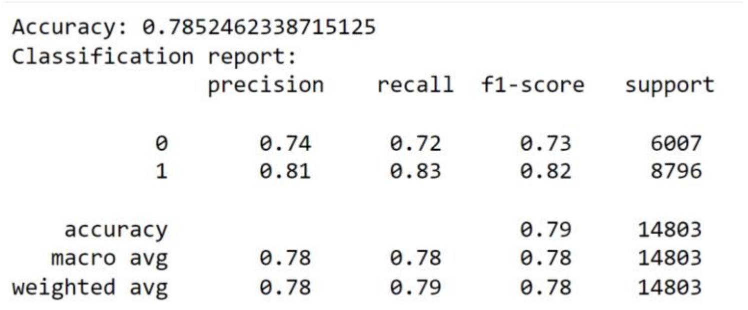


K-NN:



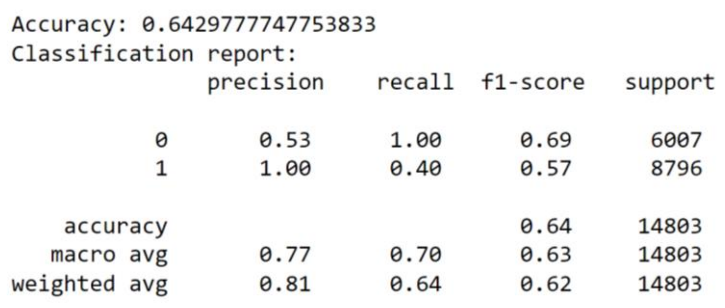
```
Accuracy: 0.8675268526650003
Classification report:
               precision    recall  f1-score   support

           0       0.83      0.84      0.84      6007
           1       0.89      0.88      0.89      8796

    accuracy                           0.87     14803
   macro avg       0.86      0.86      0.86     14803
weighted avg       0.87      0.87      0.87     14803
```

Gradient Boosting Classifier:



```
Accuracy: 0.7852462338715125
Classification report:
               precision    recall  f1-score   support

           0       0.74      0.72      0.73      6007
           1       0.81      0.83      0.82      8796

    accuracy                           0.79     14803
   macro avg       0.78      0.78      0.78     14803
weighted avg       0.78      0.79      0.78     14803
```

Gaussian Naïve Bayes Classifier:



```
Accuracy: 0.6429777747753833
Classification report:
               precision    recall  f1-score   support

           0       0.53      1.00      0.69      6007
           1       1.00      0.40      0.57      8796

    accuracy                           0.64     14803
   macro avg       0.77      0.70      0.63     14803
weighted avg       0.81      0.64      0.62     14803
```

Choosing the final model and tune the model.

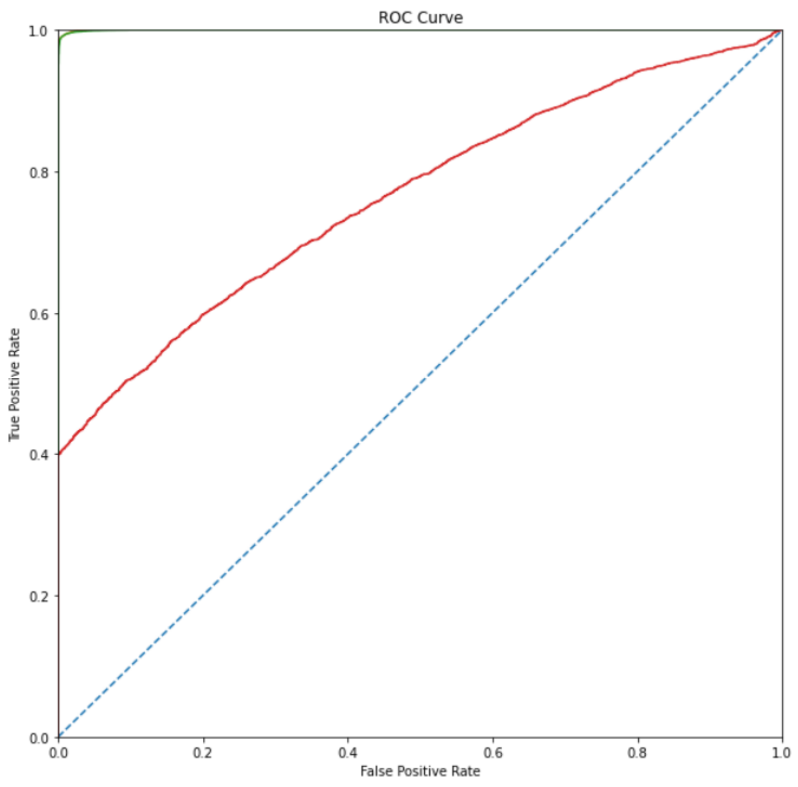
We compare the performance of three different classification models: Decision Tree, Random Forest, and Naive Bayes, using 5-fold cross-validation and ROC curve. The dataset used for this evaluation contains features related to restaurant bookings and online orders.

The output shows the average area under the receiver operating characteristic (ROC) curve (AUC) and its standard deviation for each model. The ROC curve is a graph that shows the trade-off between true positive rate (sensitivity) and false positive rate (1-specificity) for different classification thresholds. An AUC of 1 indicates a perfect classification model, while an AUC of 0.5 indicates a random guess.

According to the results, the Random Forest model performs the best with an AUC of 0.98 (+/- 0.01), followed by the Decision Tree model with an AUC of 0.90 (+/- 0.03). The Naive Bayes model has the lowest AUC of 0.76 (+/- 0.03), indicating that it performs the worst among the three models.

Overall, the results suggest that the Random Forest model is the most suitable for predicting the target variable, online order, based on the given features.

```
Decision Tree: AUC = 0.90 (+/- 0.03)
Random Forest: AUC = 0.98 (+/- 0.01)
Naive Bayes: AUC = 0.76 (+/- 0.03)
```


ROC Curve
True Positive Rate
False Positive Rate